Texture Transfer Based on Texture Descriptor Variations

Benoit Arbelot, Romain Vergne, Thomas Hurtut, Joëlle Thollot

Texture Transfer Based on Texture Descriptor Variations

Benoit Arbelot, Romain Vergne, Thomas Hurtut, Joëlle Thollot

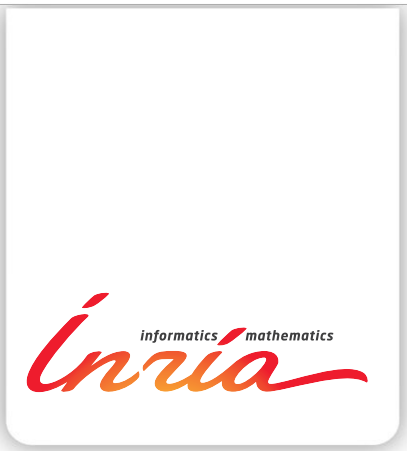
RESEARCH REPORT

N° 9067

April 2017

Project-Team Maverick

ISSN 0249-6399 ISRN INRIA/RR--9067--FR+ENG



Texture Transfer Based on Texture Descriptor Variations

Benoit Arbelot*†, Romain Vergne†, Thomas Hurtut‡, Joëlle Thollot†

Project-Team Maverick

Research Report n° 9067 — April 2017 — 38 pages

Abstract: In this report, we tackle the problem of image-space texture transfer which aims to modify an object or surface material by replacing its *input* texture by another *reference* texture. The main challenge of texture transfer is to successfully reproduce the reference texture patterns while preserving the input texture variations due to its environment such as illumination or shape variations. We propose to use a texture descriptor composed of local luminance and local gradients orientation and magnitude to characterize the input texture variations. We then introduce a guided texture synthesis algorithm to synthesize a texture resembling the reference texture with the input texture variations. The main contribution of our algorithm is its ability to locally deform the reference texture according to local texture descriptors in order to better reproduce the input texture variations. We show that our approach is able to produce results comparable with current state-of-the-art approaches but with fewer user inputs.

Key-words: texture analysis, texture transfer

* benoit.arbelot@gmail.com
† Univ. Grenoble Alpes, CNRS, Inria, France
‡ Polytechnique Montréal, Canada

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Transfert de texture basé sur des variations de descripteur

Résumé : Dans ce rapport, nous nous intéressons au transfert de texture en espace image qui consiste à modifier le matériau d'un objet ou d'une surface en remplaçant sa texture d'*entrée* par une texture de *référence*. La principale difficulté du transfert de texture est d'arriver à reproduire les motifs de la texture de référence, tout en préservant les variations de la texture d'entrée introduites par son environnement comme des variations de forme ou d'illumination. Nous proposons d'utiliser un descripteur de texture composé de la luminance locale ainsi que de l'orientation et l'amplitude locale des gradients afin de caractériser les variations de la texture d'entrée. Nous introduisons ensuite un algorithme de synthèse de texture guidé afin de synthétiser une texture ressemblant à la référence mais préservant les variations de la texture d'entrée. La principale contribution de cet algorithme est sa capacité à déformer la texture de référence localement en fonction du descripteur de texture. Cette approche permet d'obtenir des résultats comparables à l'état de l'art, mais nécessitant moins d'informations de la part de l'utilisateur.

Mots-clés : analyse de texture, transfert de texture

Contents

Figure 1: **Homogeneous textures.** These textures are statistically invariant over the image, considering a large enough window of analysis.

(a) (b) (c)

Figure 2: **Homogeneous textures in natural scenes.** Homogeneous textures may not appear homogeneous in natural scenes due to various aspects such as (a) shading variations, (b) perspective deformations or (c) support deformation.

1 Introduction

In this report, we tackle the problem of image-space texture transfer which aims to modify an object or surface material by replacing its texture by another. In the following, we call texture an image (or part of an image) that appears coherent and homogeneous at a certain scale [Mai06]. Examples of homogeneous textures are shown in Figure 1, those textures exhibit several variations due to their texture patterns, however those variations remain statistically invariant over the image.

In a natural scene however, such textures may not appear homogeneous or lose their statistical invariance due to environment changes such as shading, perspective or deformations of the texture support. Figure 2 shows examples of such cases. In the first image, shadows locally modify the grass texture luminance. In the second image, the gravel texture loses its statistical invariance due to the perspective. Finally, in the third image, the curved shape introduces shading variations and breaks the homogeneity of the texture in the image.

Texture transfer consists in replacing an *input* texture in an input image, by another *reference*, or example, texture. The reference texture can be given directly by a texture image, or extracted from a reference image. An example of texture transfer is presented in Figure 3 where the bark of a tree is transferred to another tree. Texture transfer is used in a wide range of applications such as material editing [FH04], image weathering [IEKM16, BKCO16], style transfer [FSDH16],

(a) (b) (c)

Figure 3: **Texture transfer.** The bark texture of the input tree (a) is replaced by the bark texture of the reference (b) in the final result (c). Alpha masks (bottom right) are used to define the input and reference textures in the images.

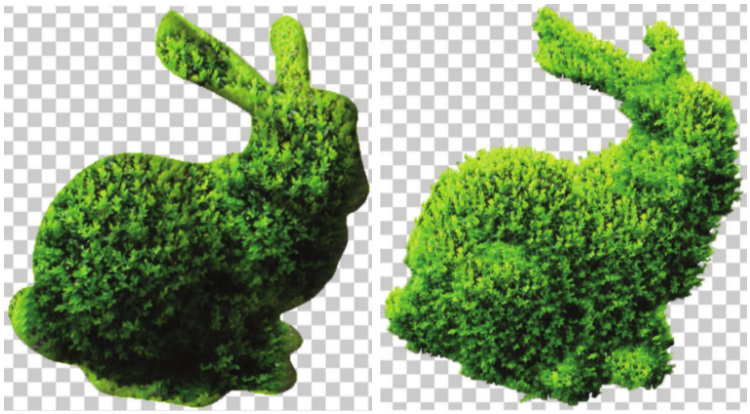
appearance modification [DBP*15] or object re-texturing [KFCO*07].

In order to accurately replace a texture in a natural image, preserving the input texture changes due to its support 3D shape and environment is crucial for a plausible result. With these considerations in mind, we make the hypothesis for our texture transfer application that the variations of a texture in a natural image are due to its own texture patterns (Figure 1) and shape and environment changes of its support (Figure 2). Under this hypothesis, the main challenge of texture transfer is to replace the input texture patterns with the reference texture patterns, while preserving the variations due to the input support shape and context.

Preserving the reference texture patterns. In order to replace the input texture with the reference texture, we use a texture synthesis algorithm based on the Image Melding algorithm [DSB*12]. This algorithm takes the reference texture as input and synthesizes over an arbitrary region a texture that is locally similar to the reference texture. To improve the synthesized result and allow extrapolation from the reference, this algorithm samples the reference texture at different scales and orientations. Since many orientations and scales can be relevant, searching through all these scales and orientations can quickly become time consuming and considerably slow the synthesis process. We believe this search can be improved by using information from texture descriptors, as detailed below. Note that in the original Image Melding algorithm, they also sampled mirrored and color corrected images, however we do not consider those search spaces as they are only relevant in very specific examples.

Preserving the input texture variations. Since extracting the shading and deformation of a 2D textured region (i.e. estimating its support 3D shape and environment) is a complex problem, many texture transfer approaches ask the user to provide these information [ZZV*03, DBP*15]. While these approaches can theoretically represent any support shape and shading, the user inputs they require are not always trivial or fast to create. To alleviate this issue, hypothesis can be made about the texture support to restrain the range of possible deformations and shading [ELS08], making the user input easier to provide.

We believe texture descriptors such as structure tensor [BvdBL*06] with edge-aware processing, as presented in [AVHT16], can help to solve both problems. The structure tensor represents



(a) (b)

Figure 4: **Effect of the alpha mask on the synthesis result.** A simple and smooth alpha mask creates unrealistic borders (a), whereas a detailed alpha mask allows for a better result (b). Images from [DBP*15].

the local compression magnitude and direction of a texture. Under the hypothesis that the texture is isotropic and uniform, then the structure tensor variations over the texture can be linked with shape and environment variations. Using this structure tensor variations to guide the texture synthesis algorithm can allow to preserve the texture variations introduced by shape and environment changes. Moreover, the local compression and orientation information provided by those descriptors can be used to automatically search the exemplar at the optimal scale and rotation by linking the compression magnitude to the texture scale, and the compression direction to the texture orientation.

Finally, since texture transfer is typically applied on a single part of the input image (corresponding to the input texture), an alpha mask is used to blend the transferred texture with the rest of the input image. An example of those masks is shown in Figure 3. Accurately computing this mask, and tuning it through the texture replacement process, is an important part of creating a plausible result when the input and reference texture showcase different features. For example when replacing a thin texture (such as skin or leather), with a texture with more thickness (such as grass or fur), the input alpha mask should be updated to reflect the reference texture features, in this case allowing fur fibers or grass blades to extend outside of the mask. An example of the effect of the alpha mask on the synthesis result is shown in Figure 4 where we can see that a simple smooth alpha mask creates unbelievable borders. To solve this, current methods typically use the information of a detailed reference mask [DBP*15, LFA*15] which captures the border complexity of the reference texture. This mask is provided by the user, but getting such detailed masks from a texture image is not always fast or easy for many users. We propose to automatically compute this detailed mask after the synthesis process, allowing the user to provide coarsely masked texture images.

In this report, we propose a new texture transfer framework which automatically provides efficient texture guides to represent deformation and shading using edge-aware texture descriptors based on color and gradients information. We use the image luminance to extract the texture local contrast, and the structure tensor of image gradients to extract the texture local compression and orientation. We apply the descriptor processing presented in [AVHT16] on these features to preserve texture edges. We also update the search process of [BSGF10] to automatically compute the best scales and orientations for every target patches, using the structure tensor eigen values

and vectors. Finally, we provide a method to automatically compute a detailed alpha mask to preserve the reference texture features from the input texture mask. In order to do so, we allow the synthesis to grow outside of the input alpha mask, then use a segmentation algorithm to make the alpha mask follow the high-frequency variations of the synthesized region. Our approach is able to create plausible texture transfer results with few user iterations: only crude alpha mask for the input and reference textures are necessary.

In summary, our contributions are the following:

- Automatically compute texture synthesis guides from textural properties.
- Automatically compute the optimal rotation and scale changes for target texture patches.
- Automatically compute a detailed alpha mask following the reference texture variations from the input alpha mask.

2 Related Works

In this section, we will first review texture synthesis and transfer methods based on user-provided information, then discuss automatic methods trying to reduce the user implication through texture analysis methods. We will also discuss style and appearance transfer methods that are close to our approach in many cases.

2.1 Texture synthesis

A complete survey of example-based texture synthesis methods was done by Wei *et al.* in [WLKT09]. Our texture synthesis algorithm belongs to the class of non-parametric synthesis methods, which includes pixel-based methods [EL99, yWL00], stitching-based methods [EF01, KSE*03, LL12], optimization-based methods [KEBK05, HZW*06, WSI07], and appearance-space texture synthesis [LH06]. Pixel-based methods synthesize the result one pixel at a time, while stitching-based methods synthesize one texture patch at a time. These methods typically fail to reproduce large scale features or patterns in the textures. Optimization-based methods solve this by synthesizing the whole texture simultaneously, and updating it until a defined criterion is met. Our texture synthesis algorithm is based on the algorithm introduced by Darabi *et al.* in [DSB*12] which unifies patch-based synthesis and texture optimization. This algorithm is based on the Patch Match algorithm [BSGF10] which computes a dense patch matching between two images quickly by alternating a random search and a propagation step. This algorithm proved fast and flexible to be adapted to various problems using texture synthesis [DBP*15, KNL*15].

2.2 Texture transfer

While some texture transfer methods rely on texture deformation [LLH04, WOBT09], many more are currently based on texture synthesis algorithms [ELS08, LJWF12, DBP*15, JFA*15] which provide the ability to extend the reference texture to an input region of arbitrary size and shape. These algorithms typically use a flat and homogeneous texture as example and produce a similarly flat and homogeneous output. While some of them can handle scale and rotation changes, they do not handle the illumination or deformation changes created by different shapes, environments or view points that we can see in real images.

In order to introduce variations in the output textures, several approaches use control maps. In [ZZV*03], Zhang *et al.* use texton masks and user-provided orientation fields and transfer functions to create progressively varying textures. Their method is well suited to handle textures

with well defined patterns such as leopard or giraffe skin, however they cannot handle complex textures such as natural foliages. Moreover, the user provided information is substantial and not easy to create. In [LH06], Lefebvre *et al.* use a user-provided guidance channel to compute a large appearance vector at each pixel, and encode this information in a low-dimensional space using principle component analysis to considerably speed-up the synthesis process. To reduce the user required data, Fang *et al.* [FH04] used a shape-from-shading technique on a textured object to estimate its normals and used these normals to guide the texture synthesis on the object. In the same vein, in [RCOL09], Rosenberger *et al.* introduced control maps to represent the non-stationarity of many real-world textures. The control maps can be user provided or automatically computed based on the assumption that the texture can be represented by several superposed layers. These superposed layers of texture represent well textures resulting from natural processes such as weathering or corrosion but lack in genericity. Extending from this idea, Lockerman *et al.* proposed automatically computed multi-scale label-maps [LSA*16] which extract clusters of different textures at different scales. Their approach is able to efficiently extract texture patches from a large panel of images. However, those approaches focus more on clustering pixels into different textures, rather than extracting the variations of a single texture.

Focusing more on textural variations due to shape or environment changes in natural images, Diamanti *et al.* proposed in [DBP*15] an object appearance manipulation framework based on the texture interpolation method of [DSB*12]. Previous texture interpolation schemes [DSB*12, PBK13, RSK13] typically compute a linear transition between two extremal texture exemplars. In these schemes, every output pixel uses the same two exemplars with different weights. The main contribution of [DBP*15] is the clustering of the exemplars using user-provided annotations, providing the ability to interpolate between different parts of the texture exemplar at each pixel. Their method is able to re-texture objects of different shapes under different lighting conditions, however they still require user-provided annotations to define the texture variations of the exemplar and the results. For complex objects, they typically extract those annotations from a 3D rendering of the object, which limit the advantage of their image-based texturing approach.

To improve the modeling of illumination and foreshortening effects, many texture transfer approaches use 3D scenes to extract relevant 3D informations and guide the 2D synthesis. In [BPLD10], Bonneel *et al.* start from a coarse user-designed 3D scene and use texture synthesis to augment it with natural textures and details. Similarly, [JDA*11] propose a data-driven approach using texture synthesis to quickly improve the realism of 3D renderings. While both approaches benefit greatly from the 3D scene information, recreating the 3D scene corresponding to an input image in order to re-texture parts of it is time consuming and not well suited for an approach which aims for user simplicity. Closer to our approach, Eisenacher *et al.* proposed in [ELS08] to transfer textures directly between photographs using texture synthesis and an approximation of the texture deformation given by the user. More specifically, the user locally describes the geometry supporting the textures by combining rational Bézier patches. However, their approach is limited to planar and cylindrical geometry for the texture support.

2.3 Texture analysis for texture synthesis

Instead of relying on user-provided information to describe the texture variations, Liu *et al.* in [LJWF12] made the assumption that many real-world textures are locally invariant. Under this assumption, low-frequency variations of the input texture are due to shading or deformation and should be preserved in the output. In practice, they linked low-frequency color variations to illumination changes and low-frequency gradient variations to deformation. In their approach they limit their deformations to scale and orientation changes. While holding true in many

cases, their assumption typically falls short when the input texture support presents sharp edges, introducing high-frequency changes in color. For example, a building corner introducing a sharp shadow on the building's texture would not be attributed to illumination of deformation changes in their framework. In [LFA*15], Lukac *et al.* took directionality and contour information into account in their texture synthesis framework. The directionality allows to accurately reproduce fibrous textures such as hairs or grass, while the contour information preserves the border effects of the texture. While these informations are useful for specific textures such as vegetation, textiles or hairs, they still rely on carefully segmented texture exemplars. Finally in [KNL*15], Kaspar *et al.* proposed to automatically extract guidance channels using the Structure Edge detector [Pio13], and compute the texture lattice to better initialize the texture synthesis. These features are well-suited to highly structured textured but do not represent well natural textures without any apparent structure such as vegetation foliages.

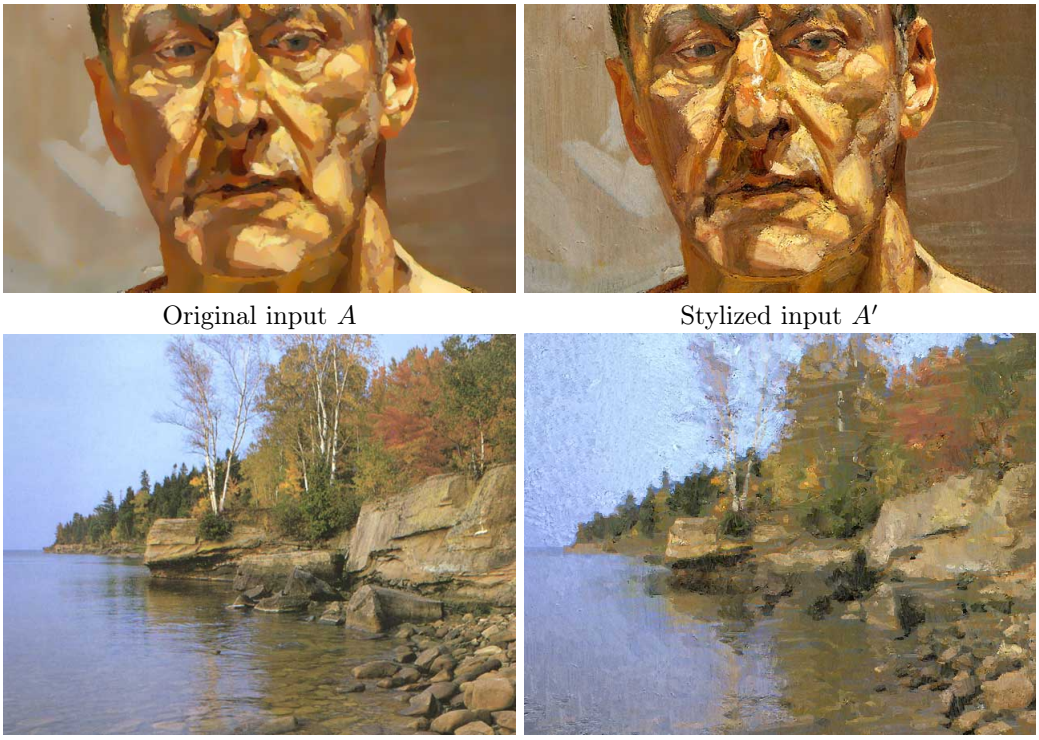
2.4 Style and appearance transfer

Style transfer typically relies on texture and/or color transfer to model the style of a reference image and apply it to an input image. Style transfer approaches can be divided in supervised and unsupervised approaches. Supervised approaches rely on a pair of reference images with two different styles. They first compute the transformation between those two images, then apply it on an input image to alter its style in a similar way. On the other hand, unsupervised approaches, similarly to our approach, rely on a single reference image and try to model its style to transfer it to an input image directly. We detail both approaches below.

One of the first supervised style transfer models the problem as computing an "image analogy" [HJO*01]. They take three images A, A' and B as input, A' being the stylized version of A, and B being the input image to be stylized into B', the output of the algorithm. The idea is that B' should relate to B in the same way that A' relates to A. They synthesize B' using pixel-based texture synthesis where a pixel of B' is selected from A' taking into account the similarity between A and B and preserving local neighborhoods from A'. An example of this approach is presented in Figure 5. This approach was extended to videos in [BCK*13]. The image analogy approach was also improved in [CVZ08] using inference on a Markov Random Field to ensure global consistency and image quilting to ensure local consistency. In a color transfer context, Shih *et al.* in [SPDF13] applied a color transformation on an image to simulate different times of day and night, where the transformation to apply is computed from a database of time-lapse videos. Color and texture transfers were combined in [OVB*15] where they apply a color transfer, then a texture transfer in areas of the image where the color transfer failed, which they determine using the reference image pair. While these approaches are efficient, they require to already have an image pair to represent the desired stylization, which can be restrictive. In our appearance frameworks, we always considered a single reference image, which would classify them as unsupervised approaches.

Unsupervised style transfer aims to alleviate the number of input needed by transferring from a single reference image directly onto the input image. In [RAF03], Rosales *et al.* used a Bayesian technique to infer the most likely output image from the input and reference images, the prior on the output image being a patch-based Markov random field obtained from the input image. Taking inspiration from the steps of drawing a picture, Zhang *et al.* in [ZCC*13] proposed to decompose the images into three additive components: draft to describe the content, paint to describe the main style, and edge to strengthened the strokes along image boundaries. Style is then transferred from the paint and edge components only, to preserve the input image content. In [SPB*14], Shih *et al.* applied style transfer between head-shots using a multiscale technique

[2] `http://www.mrl.nyu.edu/projects/image-analogies/freud.html`



Original input A | Stylized input A'

Original reference B | Stylized output B'

Figure 5: **Image analogies framework.** The transformation between A and A' is estimated and applied on B to produce B'. Images from [2].

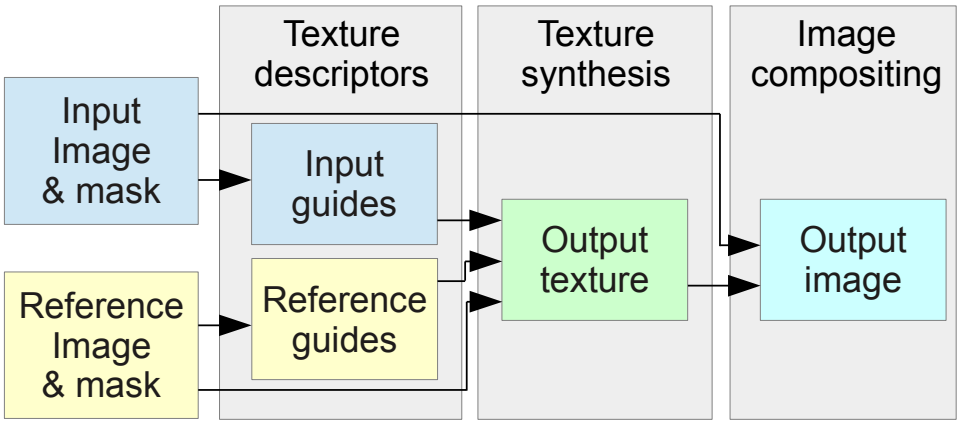


Figure 6: **Our texture transfer framework.** The texture descriptors are used to compute guides for the texture synthesis. The compositing step merges the synthesized result with the input image.

to transfer the local statistics of an example portrait onto a new one. Focusing on the textural aspect of style transfer, Frigo *et al.* in [FSDH16] transferred style between images by applying a local texture transfer on an adaptive partition of the image. Finally, Song *et al.* in [SL16] presented a similar approach to [OVB*15] by combining color and texture transfer but using a single reference image. These approaches differ from our application as they do not intend to match the style variations of the input image but rather replace it with the reference style while preserving the input structures. While we also intend to preserve the input structure, we additionally intend to match the textural variations of the input image.

Concurrently, a deep learning approach based on Convolutional Neural Networks was also used for style transfer in [GEB15] where they separate and recombine the content and style of the input and reference images. While leading to impressive results, this method is beyond the scope of our work as it relies on a pre-trained neural network architecture.

3 Our Texture Transfer Framework

Our texture transfer approach is based on texture synthesis to reproduce the reference texture, and texture descriptors to analyze the input texture variations. These descriptors allow to distinguish the variations created by the input texture patterns, which are considered statistically invariant, from the variations introduced by the texture support shape and shading. From the estimation of the variations introduced by the texture support and environment we compute guides representing those variations. Those guides are then used by our texture synthesis algorithm to synthesize the resulting texture which is locally similar to the reference texture, but still presents the variations introduced by the input texture environment.

Furthermore, when the input and/or reference texture are part of an image, alpha masks are required to composite the synthesis result with the rest of the input image. We automatically compute a detailed alpha mask for the synthesis result composition based on the reference texture variations.

Our pipeline is illustrated in Figure 6. In the following, we describe first how we estimate guides for the source and reference images using the image luminance and structure tensor in

Section 4. We present our adaptation of the Image Melding algorithm for texture synthesis in Section 5, before introducing our synthesis approach in Section 6. The synthesis results are composited into images in Section 7 and final results are presented in Section 8. Finally, discussions and future works are presented in Section 9.

4 Texture Analysis

In order to extract relevant information from the input and reference textures to create guides, we rely on texture descriptors. Many descriptors can be considered such as SIFT descriptors, Histogram of Oriented Gradient (HOG), Gabor filters, Covariance matrices, Autocorrelation features or Moment features. Lockerman *et al.* tried several descriptors to compare texture tiles in [LXDR13]. In their experiments, the moment based feature spaces outperformed autocorrelation features and HOG features. Later in [LSA*16], they compared moment based features with Gabor filters and local region statistics and did not find one descriptor to universally outperform the others. They settled on the moment features are they were computationally the simplest and fastest. Considering their analysis, we decided to use the first order moment of the texture luminance, and use the structure tensor [BG87, BvdBL*06] to describe the textures second order moments as it is fast to compute and intuitive to manipulate. In practice, the structure tensor computed on an image patch represents the principal direction of gradients (i.e. direction of highest gradient variation), and the gradient quantity in that direction (i.e. gradient compression).

Considering a neighborhood N around a pixel $\mathbf{p}$ in image I, the structure tensor $\mathbf{T}$ of $\mathbf{p}$ is computed as:

$$\mathbf{T}(\mathbf{p}) = \sum_{\mathbf{q} \in N} w(\|\mathbf{p} - \mathbf{q}\|) \nabla I(\mathbf{q}) \nabla I(\mathbf{q})^\top, \tag{1}$$

where w is a Gaussian kernel and $\nabla I(\mathbf{q})$ represents the image gradients at pixel $\mathbf{q}$. The first eigen vector $\mathbf{O}$ of $\mathbf{T}$ gives the direction of highest compression in N, while the first eigen value λ_1 gives the compression magnitude in the direction of $\mathbf{O}$. The second eigen value λ_2 gives the compression magnitude in the direction orthogonal to $\mathbf{O}$. Note that if $\lambda_1 \approx \lambda_2$, then there is no predominant compression direction in N. Furthermore, we can compute the anisotropy of the gradients inside N as $\left(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}\right)$.

Using the structure tensor information, we compute our descriptor as a 4-dimensional feature vector F defined as follows for a neighborhood N centered on pixel $\mathbf{p}$:

$$F_N(\mathbf{p}) = [L \quad \lambda_1 \quad \lambda_2 \quad \mathbf{O}],$$

where L represents the mean luminance over N computed as

$$L = \sum_{\mathbf{q} \in N} w(\mathbf{q}) L(\mathbf{q})$$

where w is a Gaussian weight. Since the direction vector $\mathbf{O}$ is normalized, it simply represents an angle.

Note that we decided not to include first and second order moments of the hue in our descriptors, although it could easily be added. We chose to discard the hue information as we found it to be necessary, on top of the luminance information, in very few cases. Doing so allowed us to keep our descriptor compact, and efficient for most of the textures we encountered.

Examples of our descriptor on several images are shown in Figure 7. We can see that λ_1 accurately gives regions of high compression in the images while $\mathbf{O}$ gives the compression direction.

λ_2 is useful to find isotropic regions (where $\lambda_1 \approx \lambda_2$) as in those regions, the direction given by $\mathbf{O}$ is not relevant. These examples also showcase the smoothing of the descriptors introduced by the neighborhood size. This smoothing is relevant when considering texture images (first three rows) as it blends related pixels (i.e. pixels from the same texture). However in the case of natural images (last two rows), it tends to smooth image silhouettes, which can introduce artefacts around those silhouettes as it blends pixels from different textures around the silhouette.

4.1 Descriptors filtering

In order to prevent the smoothing of image silhouettes due to the descriptor window on natural images, we use the multiscale gradient descent scheme presented in [AVHT16]. We apply this filtering on the luminance and structure tensor, before computing the structure tensor eigen vectors and values.

The multiscale gradient descent propagates structure tensors from homogeneous regions, where their values are accurate, to regions around image silhouettes, where the structure tensor values are skewed by the silhouette. This gradient descent is guided by the structure tensor variance computed as:

$$\mathbf{V}_N(\mathbf{p}) = \| \frac{1}{W} \sum_{\mathbf{q} \in N} (\mathbf{T}_N(\mathbf{q}) - \boldsymbol{\nu}_N)(\mathbf{T}_N(\mathbf{q}) - \boldsymbol{\nu}_N)^\top w(\mathbf{p}, \mathbf{q}) \|, \tag{2}$$

where $\mathbf{T}_N(\mathbf{p})$ is the structure tensor of pixel $\mathbf{p}$, computed over the neighborhood N, and $\boldsymbol{\nu}_N$ is the weighted average of the structure tensors over the neighborhood N.

The multiscale gradient descent is done by computing the structure tensors, their variance and applying a gradient descent along their variance at increasing scales (i.e. increasing sizes of N). We apply this on the whole structure tensor simultaneously by summing together the variances of $\mathbf{V}$ to get the variance of the structure tensor.

The effect of this multiscale gradient descent on the last image of Figure 7 is shown in the second row of Figure 8. We can observe that silhouettes are much better preserved, however undesired edges may also appear in homogeneous regions due to local minima of the structure tensors variance. In order to smooth out those edges, we apply a bilateral filtering on the multiscale gradient result, guided by the input image luminance. This filtering will smooth out edges in regions of similar luminance (homogeneous regions), while preserving silhouettes with higher contrast. The effect of this filtering is presented in the third row of Figure 8.

5 Texture Synthesis

In order to perform texture transfer, we first need a texture synthesis algorithm. Many algorithms exist as detailed in Section 2. We chose to start from the algorithm of [DSB*12] because of its speed and flexibility [DBP*15, KNL*15]. This algorithm takes as input a *reference*, or *exemplar*, and outputs a result of arbitrary dimension, locally similar to the reference. This algorithm is multiscale and starts by initializing the result with a juxtaposition of random patches from the reference image. From this it builds a Gaussian pyramid for the result and reference image and synthesizes the result from the coarsest to the finest scale. At each scale, it improves the current result iteratively by alternating two steps several times: a *search* step and a *merging* step. When the result at one scale is done, an *upsampling* step is used to get the initial result of the next scale. Each step of the algorithm is detailed below. The final result of the finest scale is the synthesis result. The pseudo-code of our texture synthesis algorithm is provided in Figure 9, while an overview of the synthesis process is shown in Figure 10. In this example, we simply

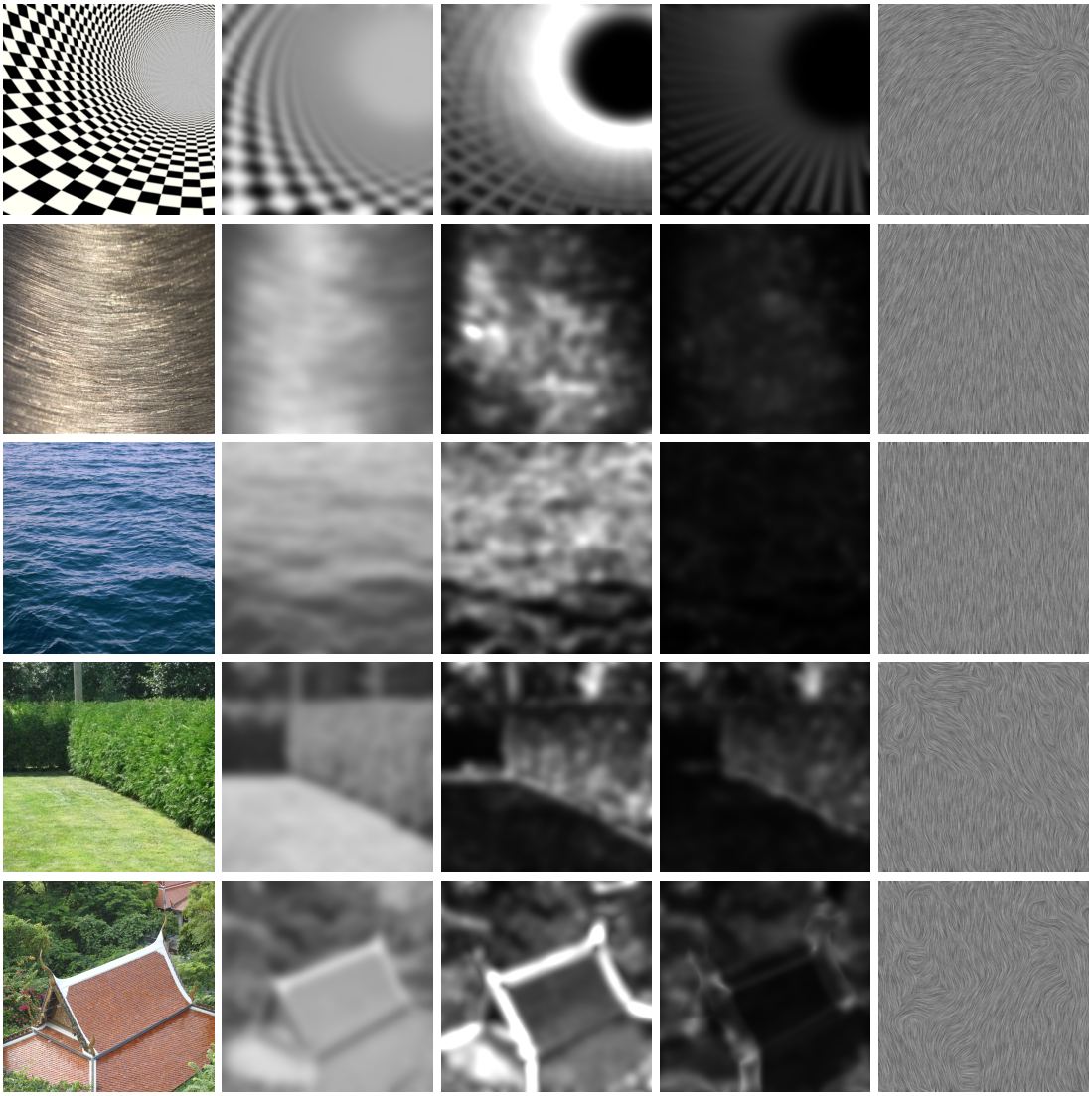


Figure 7: **Descriptor values for various texture images.** $\mathbf{O}$'s directional patterns are obtained by applying a line integral convolution [SH95] on a noise texture in the direction of $\mathbf{O}$. In these examples, $N = 30 \times 30$ and the images resolution is 512×512. Except for the first image, λ_1 and λ_2 were multiplied by 6 for better visualization.

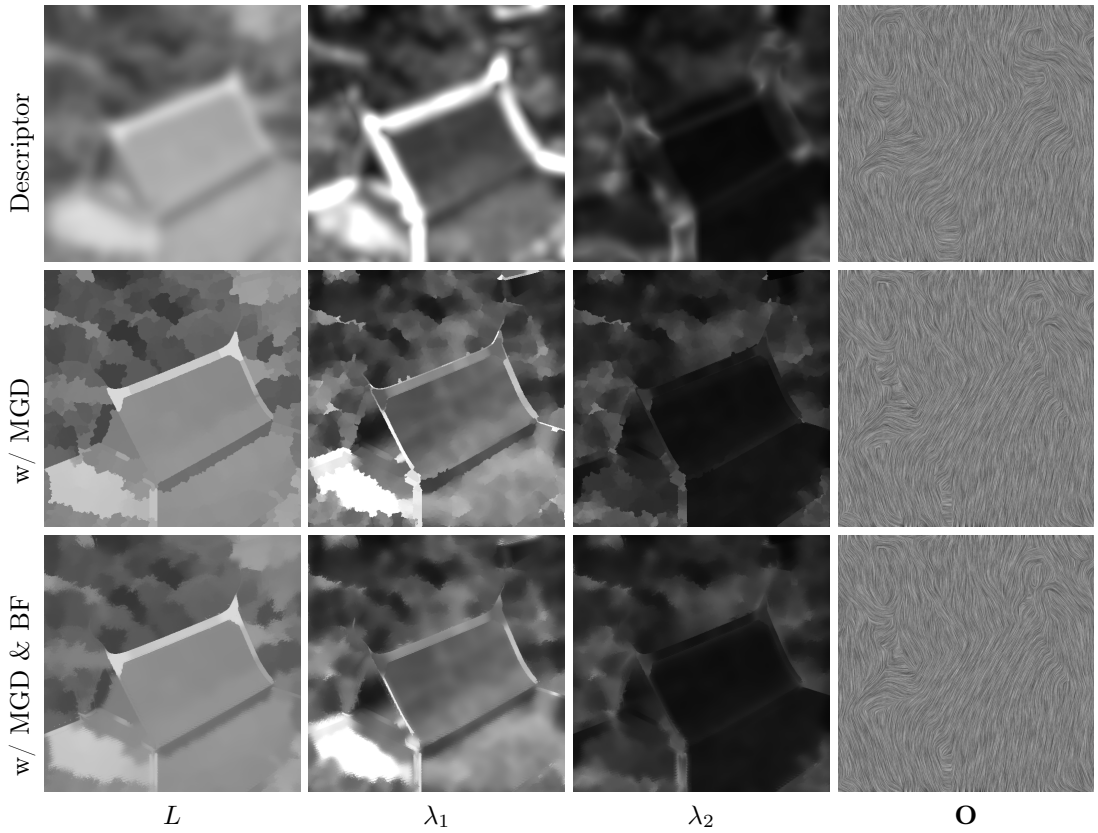


Figure 8: **Descriptor filtering.** MGD stands for Multiscale Gradient Descent, BF for Bilateral Filtering. In this example, $r_{max} = 30$ for the multiscale gradient descent, and the bilateral filtering is applied with a spatial sigma of 6, and an intensity sigma of 0.2.

Texture synthesis

Input: reference image B
Output: synthesis result R
Initialize R with random patches from B
Compute Gaussian pyramids for B and R with s_{nb} scales
for $s = s_{nb} - 1$ to 0 **do**
    if $s < s_{nb} - 1$ **then**
        $\text{NNF}_s = \text{upsample}(\text{NNF}_{s+1})$
        $R_s = \text{merging}(\text{NNF}_s)$
    end if
    for $i = 0$ to $nb_{iterations}$ **do**
        $\text{NNF}_s = \text{search}(R_s, B_s)$
        $R_s = \text{merging}(\text{NNF}_s)$
    end for
end for

Figure 9: Texture synthesis algorithm.

synthesize a result of the same size as the reference image which has a resolution of 512×512. The image pyramids are computed with 10 scales and a constant ratio between successive scales computed so that the smallest scale has a resolution of 32×32. We used a patch size of 10×10, 3 patch match iterations, and a number of synthesis iterations decreasing linearly at each scale from 25 at the coarsest scale, to 2 at the finest scale.

Search The search step is done using the Patch Match algorithm [BSGF10]. This algorithm takes as input two images A and B and outputs a *nearest-neighbor field* (NNF) which is a function $\mathbf{f} : A \mapsto \mathbb{R}^2$. This function gives for every patch coordinates $\mathbf{a}$ (location of patch centers) in A the coordinates $\mathbf{b}$ of the nearest patch in B according to a distance function D between patches. This can be seen as a patch-based optimization problem with the following energy function:

$$E(A, B) = \sum_{\mathbf{a} \in A} \min_{\mathbf{b} \in B}(D(\mathbf{a}, \mathbf{b})). \tag{3}$$

In our texture synthesis context, A is the input image and B the reference. To compute the NNF $\mathbf{f}$, the algorithm first initializes $\mathbf{f}$ with random $\mathbf{b}$ values sampled uniformly in B, then iterates two steps until convergence: a *propagation* step and a *random search* step.

The propagation step goal is to propagate good matches between neighboring pixels, based on the assumption that neighboring pixels in A will often have neighboring nearest neighbors in B. Considering a pixel $\mathbf{a}$ in A with a current nearest neighbor $\mathbf{f}(\mathbf{a}) = \mathbf{b}$, the propagation step consists in looking at the candidates $\mathbf{b}' = \mathbf{f}(\mathbf{a} - \Delta_p) + \Delta_p$ where $\Delta_p \in \{(0, 1), (1, 0), (-1, 0), (0, -1)\}$. For each of these candidates $\mathbf{b}'$, if $D(\mathbf{a}, \mathbf{b}') < D(\mathbf{a}, \mathbf{b})$, then $\mathbf{b}'$ replaces $\mathbf{b}$ as the nearest neighbor of $\mathbf{a}$ in B. This propagation step allows to quickly spread good correspondences between neighboring pixels in A, but will end up in a local minimum if used alone.

The random search step allows to avoid falling in a local minimum by sampling the reference B from an exponential distribution. Considering a pixel $\mathbf{a}$ in A with a current nearest neighbor $\mathbf{f}(\mathbf{a}) = \mathbf{b}$, the random search step will look at candidates $\mathbf{b}' = \mathbf{b} + m\alpha^i \mathbf{R}_i$, where $\mathbf{R}_i$ is a random value in $[-1, 1] \times [-1, 1]$, m is the maximum image dimension, and α is a ratio between window sizes, usually set to $\alpha = 1/2$. The index i is increased from $i = 0, 1, 2, ..., n$ until the search

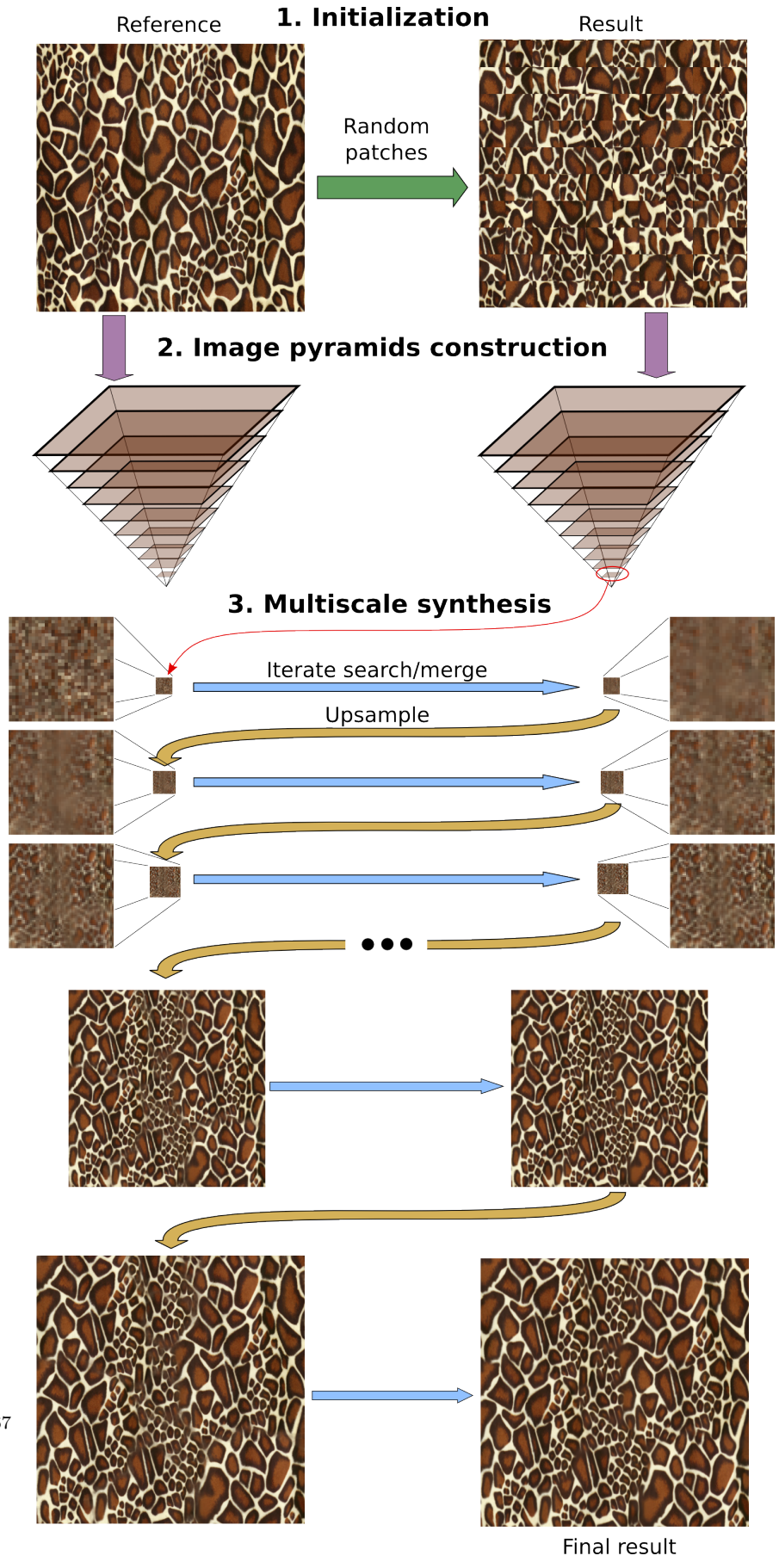


Figure 10: **Texture synthesis algorithm.** See text for details.

Patch Match

Input: input image A and reference image B
Output: nearest neighbor field $\mathbf{f}$
Initialize $\mathbf{f}$ with random coordinates of B
for $i = 0$ to $nb_{iterations}$ **do**
  for every pixel $\mathbf{a}$ of A **do**
    $\mathbf{f}(\mathbf{a}) = \text{Propagation}(\mathbf{f}, \mathbf{a})$
    $\mathbf{f}(\mathbf{a}) = \text{RandomSearch}(\mathbf{f}, \mathbf{a})$
  end for
end for

Propagation

Input: current NNF $\mathbf{f}$ and pixel $\mathbf{a}$
Output: updated NNF $\mathbf{f}'$
for $\Delta_p \in \{(0,1),(1,0),(-1,0),(0,-1)\}$ **do**
  if $D(\mathbf{a}, \mathbf{f}(\mathbf{a}-\Delta_p)+\Delta_p) < D(\mathbf{a}, \mathbf{f}(\mathbf{a}))$ **then**
    $\mathbf{f}'(\mathbf{a}) = \mathbf{f}(\mathbf{a}-\Delta_p)+\Delta_p$
  end if
end for

RandomSearch

Input: current NNF $\mathbf{f}$ and pixel $\mathbf{a}$
Output: updated NNF $\mathbf{f}'$
$i = 0$
while $m\alpha^i > 1$ **do**
  if $D(\mathbf{a}, \mathbf{f}(\mathbf{a}) + m\alpha^i \mathbf{R}_i) < D(\mathbf{a}, \mathbf{f}(\mathbf{a}))$ **then**
    $\mathbf{f}'(\mathbf{a}) = \mathbf{f}(\mathbf{a}) + m\alpha^i \mathbf{R}_i$
  end if
  $i = i + 1$
end while

Figure 11: **Patch Match algorithm.**

radius $m\alpha^i$ is below 1 pixel. The pseudo-code of the Patch Match algorithm is given in Figure 11.

We use a distance D combining two terms: the first one is the Euclidean distance between the patches colors in the CIELab color space, the second one is an occurrence term as in [KNL*15]. The first term allows to find patches that have similar color distributions. The second term enforces a uniform sampling of the reference image to avoid oversampling reference patches that are local minima in the distance space. Mathematically, the distance between two patches centered on pixels $\mathbf{a}$ and $\mathbf{b}$ in A and B is computed as follows:

$$D(\mathbf{a},\mathbf{b}) = \lambda_{\text{col}} \sum_{\mathbf{x}\in\mathbf{P}} \|A(\mathbf{a}+\mathbf{x}) - B(\mathbf{b}+\mathbf{x})\|^2 + \lambda_{\text{occ}} \sum_{\mathbf{x}\in\mathbf{P}} \frac{\Omega(\mathbf{b}+\mathbf{x})}{h^2 \omega_{\text{best}}} \tag{4}$$

where $\mathbf{P}$ represents the patch coordinates in $[-h/2, h/2] \times [-h/2, h/2]$ with h as the patch height and width. λ_{col} and λ_{occ} control the influence of each term in the distance computation. $\Omega(\mathbf{b})$

(a) Reference (b) $\lambda_{\text{occ}} = 0$ (c) $\lambda_{\text{occ}} = 0.01$ (d) $\lambda_{\text{occ}} = 0.1$

Figure 12: **Influence of the occurrence term.** Without occurrence (b), the reference (a) is only partially sampled. The purple area especially is over-sampled and reproduced several time in the result. When the occurrence is added (c) more areas of the reference are used and the red region for example appears in the result. Increasing the occurrence weight (d) samples more and more of the reference.

refers to the number of occurrence of pixel $\mathbf{b}$, i.e. the number of pixel in A that have $\mathbf{b}$ as the nearest neighbor in B, and ω_{best} represents the occurrence value for a uniformly sampled reference, defined as

$$\omega_{\text{best}} = \frac{|A|}{|B|} h^2, \tag{5}$$

where $|A|$ and $|B|$ represent the resolutions of the input and reference respectively. The influence of the occurrence on a synthesis result is presented in Figure 12. In this example the purple region of the reference is a local distance minimum and is over-sampled without occurrence. Increasing the influence term weight allows to sample the reference more uniformly and limit the amount of pattern repetitions in the result.

Merging From the search step, we have the NNF between the current result and the reference image. Since the NNF was computed using image patches of size $h \times h$, every pixel $\mathbf{a}$ in the current result A have $h \times h$ candidates in the reference image B, one from each patch $\mathbf{P}$ overlapping $\mathbf{a}$. The original Image Melding algorithm simply averages those candidates to get the final color of pixel $\mathbf{a}$ in the new result R:

$$R(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbf{P}} \frac{B(\text{NNF}(\mathbf{a} - \mathbf{x}) + \mathbf{x})}{h^2}. \tag{6}$$

Since the mean is strongly impacted by outliers we weight each candidate by the inverse of its patch distance computed during the search. Furthermore, we apply an additional weight based on a Gaussian fall-off function so that candidates closer to the center of their patch will have more weight, similarly to [KEBK05]. Our final mean is computed as follows:

$$R(\mathbf{a}) = \frac{1}{W} \sum_{\mathbf{x} \in \mathbf{P}} \frac{w(\|\mathbf{x}\|) B(\text{NNF}(\mathbf{a} - \mathbf{x}) + \mathbf{x})}{D(\mathbf{a} - \mathbf{x}, \text{NNF}(\mathbf{a} - \mathbf{x}))}, \tag{7}$$

with

$$W = \sum_{\mathbf{x} \in \mathbf{P}} \frac{w(\|\mathbf{x}\|)}{D(\mathbf{a} - \mathbf{x}, \text{NNF}(\mathbf{a} - \mathbf{x}))}. \tag{8}$$

Upsampling To get from a scale to the next finer one, we need to upsample the current scale result. In this case, directly upsampling the result image is not ideal as it would not re-introduce the fine scale details missing from the coarse scales. To solve this, as in [WSI07], the final NNF is upsampled instead, before updating the patch distances and merging the higher resolution patches directly to re-introduce fine details from the higher scale.

6 Guided Texture Synthesis

For the texture transfer application, the synthesized result should not only resemble the reference, but also follow the variations of an input texture. This input texture variations are described by guidance images, also called guides or feature channels [ELS08, DBP*15, KNL*15]. Guides are computed on the input and reference image and used by the texture synthesis algorithm at the search step to find corresponding patches between the current result and the reference. More specifically, the input and reference guides differences are added in the patch distance computation (Equation 4).

6.1 Texture guides

We use the descriptors presented in Section 4 to automatically compute the guides for the input and reference images. These descriptors provide guides representing the texture local luminance, compression and direction. We describe the integration and effect of each of these guides below.

Luminance guides The luminance guides L are simply added to the patch distance D of Equation 4 as the Euclidean distance between the patches luminance values:

$$D'(\mathbf{a},\mathbf{b}) = D(\mathbf{a},\mathbf{b}) + \lambda_{\text{lum}} \sum_{\mathbf{x}\in\mathbf{P}} \|L_A(\mathbf{a}+\mathbf{x}) - L_B(\mathbf{b}+\mathbf{x})\|^2 \tag{9}$$

where λ_{lum} controls the luminance guides influence.

The effect of the luminance guides is shown in Figure 13 where the input luminance is better preserved with high values of λ_{lum}, but the reference texture is badly preserved. The extreme case of only using the guides, Figure 13 (b), highlights the importance of the color term in the patch distance to preserve the small scale variations of the reference.

Compression guides The texture compression guides $\mathbf{C}$ combine the eigen values of the structure tensor, $\mathbf{C}(\mathbf{a}) = (\lambda_1(\mathbf{a}), \lambda_2(\mathbf{a}))$. Once again they are added to the patch distance D as the Euclidean distance between the patches compression values:

$$D''(\mathbf{a},\mathbf{b}) = D(\mathbf{a},\mathbf{b}) + \lambda_{\text{comp}} \sum_{\mathbf{x}\in\mathbf{P}} \|\mathbf{C}_A(\mathbf{a}+\mathbf{x}) - \mathbf{C}_B(\mathbf{b}+\mathbf{x})\|^2 \tag{10}$$

where λ_{comp} controls the compression guides influence.

The effect of the compression guides is shown in Figure 14 where the perspective from the input is restored in the result with the addition of the texture compression guides.

Direction guides The texture direction guides $\mathbf{O}$ contain the first eigen vector of the structure tensor, $\mathbf{O}(\mathbf{a}) = (O_x(\mathbf{a}), O_y(\mathbf{a}))$. Since this eigen vector is normalized, the direction distance is added to the patch distance as follows:

$$D'''(\mathbf{a},\mathbf{b}) = D(\mathbf{a},\mathbf{b}) + \lambda_{\text{dir}} \sum_{\mathbf{x}\in\mathbf{P}} 1 - |\mathbf{O}_A(\mathbf{a}+\mathbf{x}) \cdot \mathbf{O}_B(\mathbf{b}+\mathbf{x})| \tag{11}$$

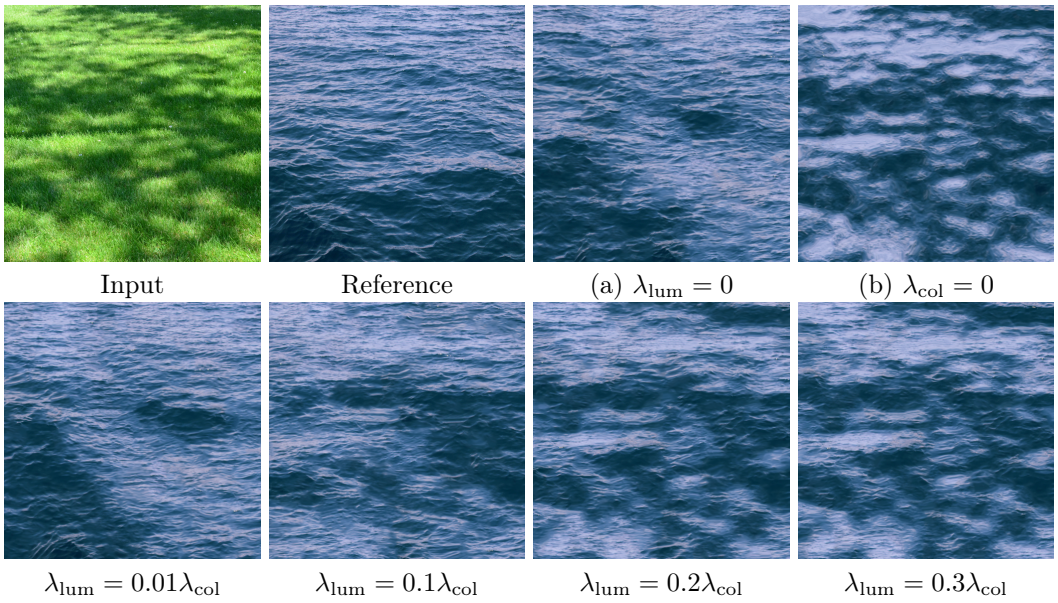


Figure 13: **Influence of the luminance guides.** On the top row, with the color distance only (a), the result luminance is not constrained. With the luminance guides only (b), the reference texture is badly preserved. On the second row, the combination of both (with $\lambda_{\text{col}} = 1$) allows to preserve the reference texture while following the input luminance. The higher λ_{lum}, the closer the result luminance is to the input luminance. In those examples, $\lambda_{\text{occ}} = 0.1$.

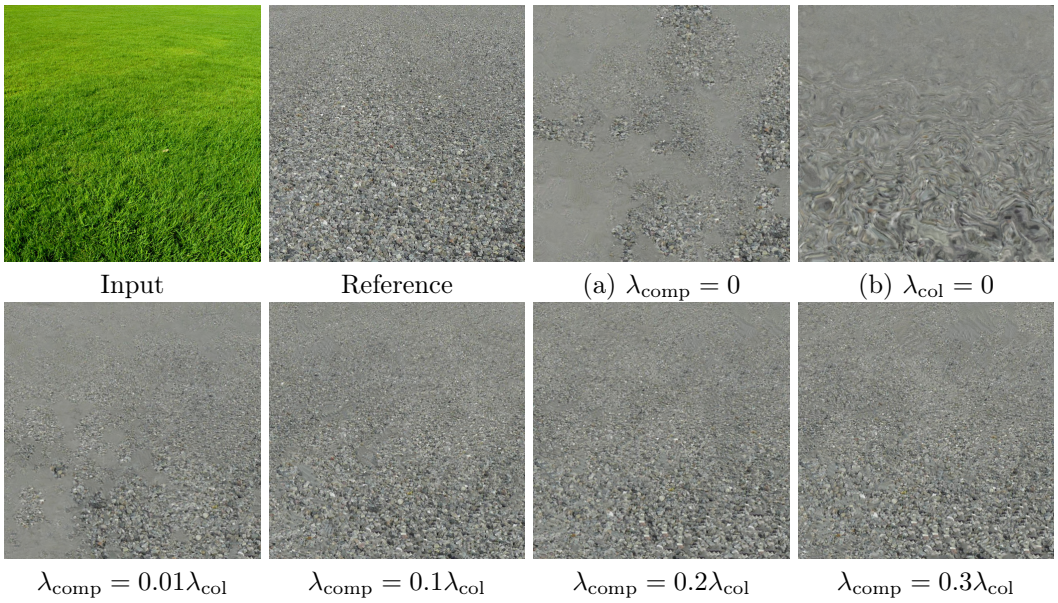


Figure 14: **Influence of the texture compression guides.** On the top row, without guides (a) the result loses the input perspective. With the guides only (b), the reference texture is badly preserved. On the second row, the combination of both (with $\lambda_{\text{col}} = 1$) allows to preserve the reference texture while following the input compression. In those examples, $\lambda_{\text{occ}} = 0.1$.

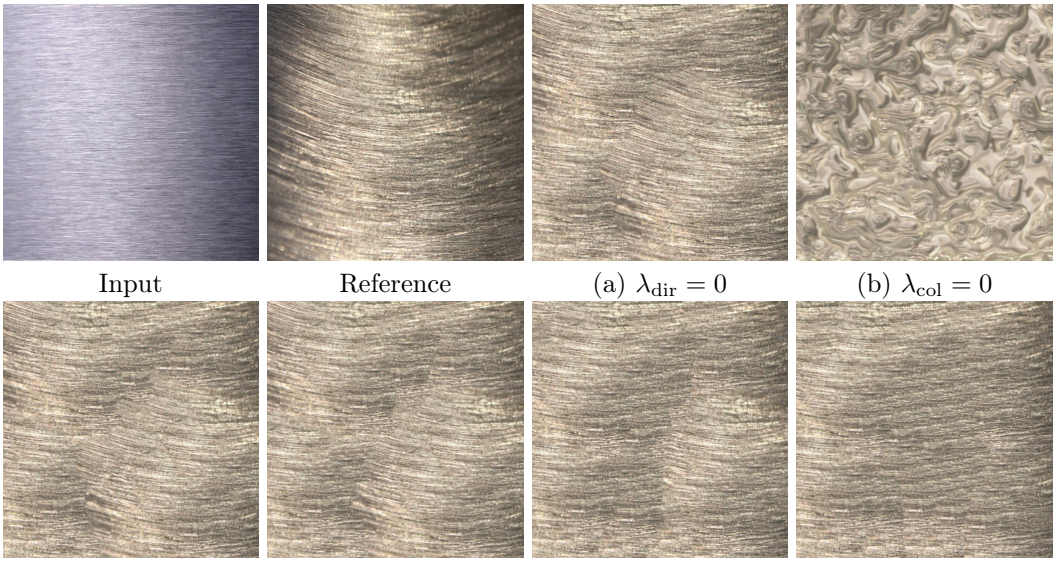


Figure 15: **Influence of the texture direction guides.** On the top row, without guides (a) the result loses the input scratches direction. With the guides only (b), the reference texture is badly preserved. On the second row, the combination of both (with $\lambda_{col} = 1$) allows to preserve the reference texture while getting closer to the input direction. In those examples, $\lambda_{occ} = 0$ since we do not intend to sample the whole reference, but rather only the patches with good direction.

where λ_{dir} controls the direction guides influence.

The effect of the direction guides is shown in Figure 15. In this example, the direction guides are used to sample only patches in the reference that have the same direction as the input. The result with a high value of λ_{dir} brings the result direction closer to the input direction, even though very few patches in the reference are perfectly oriented in the same direction as the input.

Those guides allow to accurately sample the reference in order to preserve the luminance, texture compression and direction of the input. However, as showcased in the direction guide example, Figure 15, sometimes the reference contains only few patches of similar textural properties as the input. In those cases, a solution is to deform exemplar patches by rotating and scaling them and adjusting their luminance in order to better match input patches.

6.2 Exemplar deformation

In order to extend the range of possible patches from the exemplar, search amongst different scales and rotations is often used [BSGF10, DSB*12, DBP*15]. To simulate the rotation and scaling of patches, these methods simply use more exemplar images corresponding to rotated and scaled version of the original exemplar, then search in all those exemplars with the Patch Match algorithm. While this is simple, it has two main drawbacks. First, the search time is increased due to the increase in the search space size. Second, it only allows pre-defined rotations and scaling of patches. Consequently, increasing the number of possible rotations or scaling increases the number of exemplars to explore during the search. To alleviate this issue, we propose to

(a) Input & Ref. (b) luminance guides (c) luminance alignment

Figure 16: **Luminance alignment.** Since the reference luminance range is lower than the input (a), the luminance guides (b) cannot reproduce the input luminance and over-sample the regions of high luminance of the reference. Luminance alignment (c) allows to change the contrast of reference patches in order to better follow the luminance variations of the input.

use the descriptor information, more specifically the texture compression and direction guides to automatically align candidate patches with the input patch during the search. This allows to compute a specific orientation and scaling for each patch, while only searching the original exemplar. Moreover, we can also use the luminance guides information to adjust the contrast of the reference patches before matching. The details of each alignment is given below.

Luminance alignment Consider two pixels $\mathbf{a}$ in the input image A and $\mathbf{b}$ in the exemplar image B whose local luminance means are given by $L^A(\mathbf{a})$ and $L^B(\mathbf{b})$ respectively. We simply bring the mean luminance of the reference patch $L(\mathbf{b})$ to the mean luminance of the input patch $L(\mathbf{a})$ by updating the reference patch luminance values $B_l(\mathbf{b})$ before computing the color distance:

$$B'_l(\mathbf{b}) = B_l(\mathbf{b}) - (L(\mathbf{b}) - L(\mathbf{a})). \tag{12}$$

The luminance difference $L(\mathbf{b}) - L(\mathbf{a})$ of the best patch is stored in the NNF in order to apply it again on the patch in the merging step. An example of luminance alignment compared to the luminance guides is given in Figure 16. In this example, the input has a greater luminance range than the reference, making the luminance guides oversample the region of high luminance of the reference. Using the luminance alignment, we are able to better follow the input luminance while sampling the reference more evenly, reducing the amount of repetitions in the result.

Anisotropic scale alignment Consider two pixels $\mathbf{a}$ in the input image A and $\mathbf{b}$ in the exemplar image B whose compressions are given by $\boldsymbol{\lambda}^A_{1,2}(\mathbf{a})$ and $\boldsymbol{\lambda}^B_{1,2}(\mathbf{b})$ respectively. We first normalize each compression guide by dividing it by its maximum value in order to bring the input and reference compressions in similar ranges. Then we compute the anisotropic scaling $\mathbf{S}$ to apply to the patch as the ratio between the input and reference patches compressions:

$$\mathbf{S} = \frac{\boldsymbol{\lambda}(\mathbf{a})}{\boldsymbol{\lambda}(\mathbf{b})}. \tag{13}$$

To avoid divergences when compressions vary to much, we clamp this ratio to keep it in the interval $[\mathbf{S}_{\min}, \mathbf{S}_{\max}]$. Typically, we use $\mathbf{S}_{\min} = (0.5, 0.5)$ and $\mathbf{S}_{\max} = (2, 2)$ to avoid excessive

(a) Input & Ref. (b) compression guides (c) scale alignment

Figure 17: **Anisotropic scale alignment.** Since the reference (a) has a single scale in it, the compression guides (b) cannot reproduce the scale change of the input. Scale alignment (c) allows to deform the reference patches in order to follow the compression variations of the input.

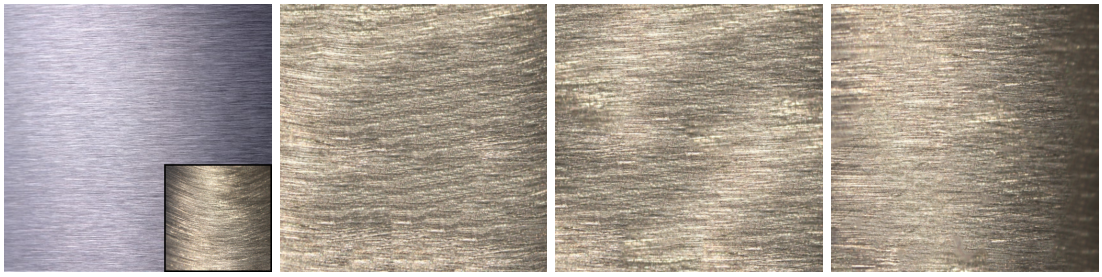
distortions of the reference patches. The scaling $\mathbf{S}$ of the best matching patch is stored in the NNF in order to apply the same scaling on the exemplar before the merging step. An example of scale alignment compared to the compression guides is given in Figure 17. We can see in this example that the compression guides fail to reproduce the scale variation of the input with the single scale of the reference. Scale alignment allows to deform the reference patches to better follow the input scale variations.

Rotation alignment Consider two pixels $\mathbf{a}$ in the input image A and $\mathbf{b}$ in the exemplar image B whose directions are given by $\mathbf{O}^A(\mathbf{a})$ and $\mathbf{O}^B(\mathbf{b})$ respectively. From the direction $\mathbf{O}$, we derive the rotation angle $\Theta = \text{atan}(O_x/O_y)$. We then apply a rotation on the exemplar, centered on $\mathbf{b}$, of angle:

$$\Theta_{\text{diff}} = \delta\Pi + \Theta(\mathbf{a}) - \Theta(\mathbf{b}) \tag{14}$$

where $\delta \in \{0, 1\}$ allows to choose between the two opposite orientations corresponding to the same direction. In practice, we compare the input patches with both candidates ($\delta = 0$ and $\delta = 1$). Afterwards, we store the rotation Θ_{diff} of the best matching patch in the NNF in order to apply the same rotation on the exemplar before the merging step. An example of result with this rotation alignment is presented in Figure 18. This example shows that rotation alignment of patches before comparison allows to more evenly sample the exemplar, avoiding repetitions and better matching the input direction. Adding the luminance guides allows to get a resulting texture preserving the input texture characteristics.

These alignment strategies are useful to simulate reference variations when the original reference contains less variations than the input. However, the deformation of the reference patches also distort the reference texture, which can deteriorate the results. To avoid needless deformations of the reference patches, we favor using the guides when the input and reference present similar textural variations, and align the reference patches when the reference has less textural variations than the input. This is currently done manually by choosing which guide and alignment to use. We plan to do it automatically in future work by comparing the range of textural variations in the input and reference images. We can also combine guides for a feature, and alignment on another to better use the reference texture. An example of this combination is shown in Figure 18 (d) where we used the luminance guides since the input and reference have



(a) Input & Reference | (b) direction guides | (c) rotation alignment | (d) rotation align. & lum. guides

Figure 18: **Rotation alignment.** Direction guides (b) sample only the few well-oriented patches of the reference, introducing many repetitions. Rotation alignment (c) allows to more evenly sample the exemplar by aligning reference patches with the input direction before comparison. Adding the luminance guide (d), we get a result that preserves well the input texture variations.

similar luminance variations, and rotation alignment since they do not have similar direction variations.

7 Image Compositing

In order to replace a textured region in an image, an alpha mask delimiting this region is needed. This mask is defined by the user and can be created manually, or through the use of strokes and texture descriptors as described in [AVHT17]. This mask also provides a silhouette information that can be used as a guide, and can be updated during the synthesis for better results with specific textures as described below.

7.1 Mask guides

The mask guides M are added to the patch distance D of Equation 4 as the Euclidean distance between the patches mask values:

$$D''''(\mathbf{a},\mathbf{b}) = D(\mathbf{a},\mathbf{b}) + \lambda_{\text{mask}} \sum_{\mathbf{x}\in\mathbf{P}} \|M_A(\mathbf{a}+\mathbf{x}) - M_B(\mathbf{b}+\mathbf{x})\|^2 \tag{15}$$

where λ_{mask} controls the mask guides influence.

An example of synthesis using the mask guides is presented in Figure 19 where the patterns and collar of the reference pull are transferred to the input thanks to the silhouette information.

7.2 Mask expansion

In the example of Figure 19, the objects masks are fairly simple and constant through the synthesis because the synthesized texture should not change the object silhouette. Changing the bark of the input tree should keep the tree silhouette mostly unchanged. However, in cases where the reference texture present several micro-patterns with semantic significance, leaves in a foliage texture for example, those patterns may alter the silhouette (i.e. the mask) of the input. When not taken into account during the synthesis, i.e. when the input mask is left unchanged, the clamping of those patterns lead to very unnatural results, as seen in Figure 21 without the

Input with mask | Reference with mask | Result

Figure 19: **Synthesis with mask guides.** The mask guides ensure that the silhouettes of the result are synthesized with corresponding silhouettes of the reference region. In those examples, $\lambda_{\text{col}} = 1$, $\lambda_{\text{occ}} = 0.01$ and $\lambda_{\text{mask}} = 1$.

mask expansion. To solve this, previous methods usually rely on a detailed reference mask which is used to synthesize the result mask [LFA*15, DBP*15]. Since creating this detailed mask is not always fast or simple, we propose to compute it automatically from the coarse input mask and the synthesis result.

In order to create a refined result mask, we first dilate the input mask M_A with morphological dilatation to get an new input mask M'_A. We then synthesize the texture inside this expanded mask in order to get texture information outside of the original object silhouettes. Using the synthesis result, we expand M_A by iteratively adding to the mask pixels whose color is close to their neighbors color. The color distance D_{col} between two pixels is computed as the Euclidean distance in the Lab color space and we use a threshold τ_c under which pixels are added to the mask. To remove noise from the result texture patterns, we also apply a bilateral filter on the synthesized texture before computing the mask expansion. The pseudo-code of this expansion is given in Figure 20, and an example of result is shown in Figure 21. We can see in this example that expanding the input mask to follow the synthesized foliage produce a much better result than keeping the input mask. Furthermore, the complex result mask is computed automatically from the coarse input mask.

8 Results

8.1 Implementation & performances

With input images of resolution 512×512, we set the texture synthesis parameters as follows for most of our results:

- **Image pyramids.** We use image pyramids with 10 scales, the lowest scale resolution is set to 32×32 and the intermediate scale resolutions are computed to have a constant ratio between scales.
- **Number of synthesis iterations per scale.** We use a number of iterations linearly decreasing from 12, at the coarsest scale, to 1, at the finest scale. Since the algorithm slowly converges through each scale, less and less iterations are needed per scale.

Mask expansion

Input: synthesis result R, input mask M_A
Output: expanded mask M_R
Initialize M_R with the input mask M_A
for $i = 0$ to $nb_{iterations}$ **do**
    for every pixel $\mathbf{p} \notin M_R$ **do**
        if a neighbor $\mathbf{q}$ of $\mathbf{p}$ is in M_R **and**
          $D_{col}(\mathbf{p}, \mathbf{q}) < \tau_c$ **then**
            Add $\mathbf{p}$ to M_R
        end if
    end for
end for

Figure 20: **Mask expansion algorithm.**

Input Mask | Reference | w/o expansion | w/ expansion

Figure 21: **Mask expansion from the synthesis result.** We automatically expand the simple input mask in order to alter the result silhouette according to the reference micro-structures.

- **Number of Patch Match iterations.** We use only 3 Patch Match iterations since the algorithm converges quickly and is used at every synthesis iteration.
- **Patch size.** We use a patch size of 10×10 in all results unless specified otherwise. This large patch size allows to better capture texture patterns, while the multiscale approach keeps the synthesis relatively fast.
- **Guide weights.** In every result, we keep $\lambda_{\text{col}} = 1$ and adjust the weight of the guides and occurrence through their respective λ values.

Our algorithm is fully implemented on the GPU. The propagation of the Patch Match algorithm is done in parallel using an adaptation of the jump flood scheme of [RT06], as in [BSFG09]. With the above mentioned parameters, we synthesize a 512×512 result in approximately 40 seconds on a GeForce GTX 670 graphic card.

8.2 Results and discussion

This section presents results for texture transfer between textures in Figure 22, between input images and reference textures in Figure 23, and between images in Figure 24. Everytime a mask is used, $\lambda_{\text{mask}} = 1$. These results show that different guides and/or alignments can be

used, depending on the desired result. Luminance alignment allows to preserve the input texture luminance in the second and first row of Figure 23 and 24 respectively. Rotation alignment allows to preserve the scratches direction in the first row of Figure 22, the water splashes direction in the first row of Figure 23, and the bark direction in the second row of Figure 24.

Scale alignment needs a special attention as linking the compression information to the perceived image scale is not always straightforward. The equation 13 links coarse scales to low compression, and fine scale to high compression. However this correspondence may not accurately reflect the perceived scales in an image. For example, when considering perspective images, as the input of the second row of Figure 22, the far part of the texture contains less compression because several far texture patterns are blurred inside each pixel. In this case, those low compressions do not represent coarser scales, but actually extremely fine scales whose details are blurred. To account for this effect in perspective images, we inverse the scale extracted from the scale alignment in Equation 13. The result of this is shown in the second row of Figure 22 where the reference patches are stretched in the foreground where the compression is high, and compressed in the background where the compression is low, leading to a better perspective reproduction. To better illustrate this phenomenon, we show the compression of a synthetic example with an ever increasing scale in Figure 25. As we can see in this example, at the coarser scales (foreground), scale increments are matched with an increasing compression. However a limit is reached at the red circle where the scale increase starts bringing several gradients inside a single pixel. At this point, the compression stops increasing and decreases back to zero as the scale goes to infinity. In this type of example, additional information is needed to recognize that the increase and decrease of compression actually represent a monotonously varying image scale.

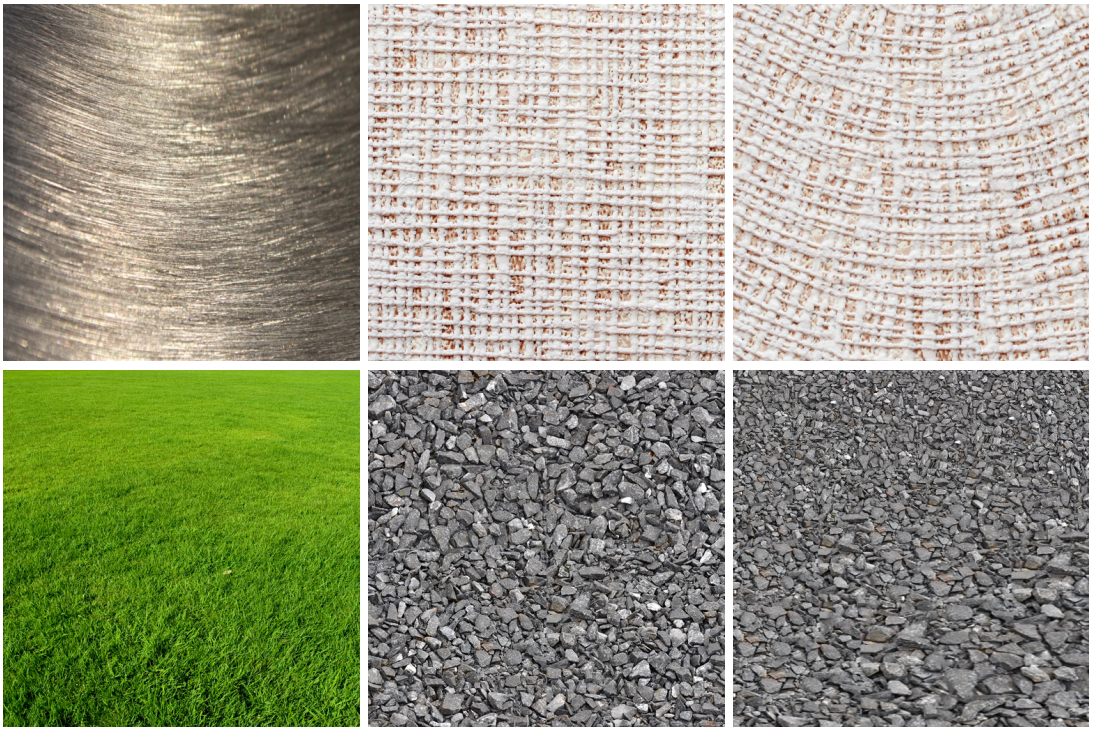
Our approach is similar to [LJWF12] as we also use texture descriptors to guide texture synthesis. However their guides are computed with standard texture descriptors, introducing blurriness and halos in their result due to the descriptor window of analysis. Our descriptor after edge-aware descriptor filtering allows to get sharper guides around texture transitions, as shown in Figure 26.

We use a similar texture synthesis algorithm as [DBP*15] including guidance channels. However the guides used in [DBP*15] have to be entirely provided by the user. In comparison, we automatically compute guides representing texture variations and only need coarse alpha masks to be provided by the user. Since their synthesis algorithm differs from ours as it focuses more on interpolation between different reference regions, we could combine their synthesis algorithm with our guides to reduce their number of user inputs.

Moreover, most of existing texture synthesis methods need additional reference images to integrate exemplar deformation. Each additional reference represents a specific deformation, a specific scaling or rotation for example. Our patch alignment strategy based on our texture description allows us to automatically choose the optimal anisotropic scaling, rotation and luminance change for an exemplar patch to fit the corresponding input patch.

9 Conclusion & Limitations

In this report, we proposed to use texture descriptors to characterize texture variations and from these, guide a texture synthesis algorithm to transfer the variations of the input texture to a reference texture. These guides represent the texture local luminance, compression and direction. They are computed using the luminance and structure tensor, followed by the edge-aware descriptor filtering presented in [AVHT16]. We use these guides in two ways. When the input and reference textures have similar texture variations, we use those guides to restrict the patch selection in the reference to patches of similar texture variations as the input. When the



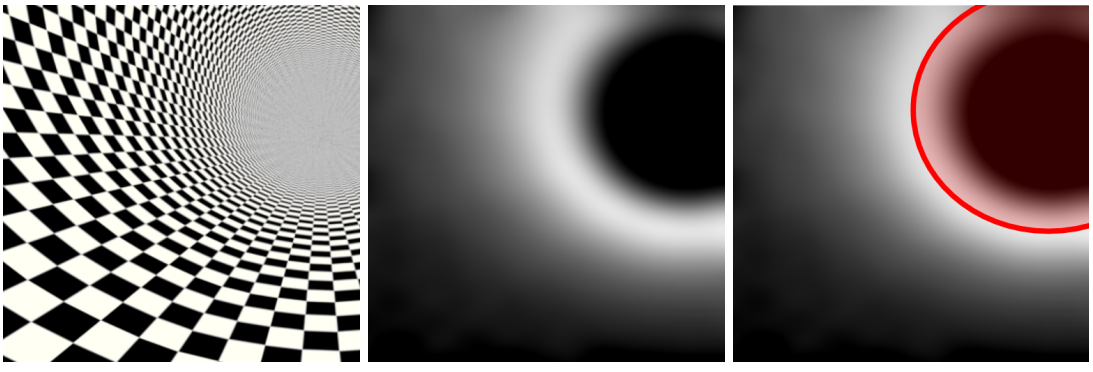
Input Reference Result

Figure 22: **Transfers between textures.** First row: rotation alignment. Second row: scale alignment.

Figure 23: **Transfers between image and texture.** First row: rotation alignment. Second row: luminance alignment with $\lambda_{\text{occ}} = 0.1$.

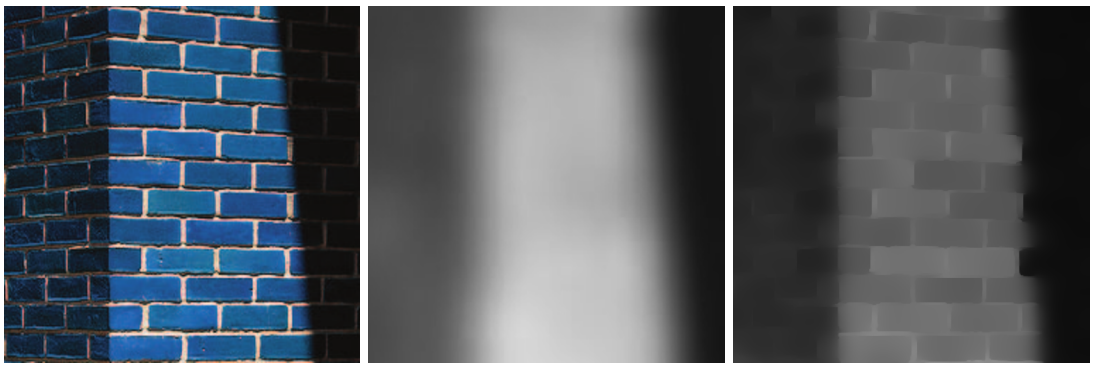
Input Reference Result

Figure 24: **Transfers between images.** First row: luminance alignment with $\lambda_{\text{occ}} = 0.1$. Second row: rotation alignment with $\lambda_{\text{occ}} = 0.01$.



Input Compression Inversion highlighted

Figure 25: **Limitation of scale description with compression.** This synthetic example has an increasing scale from the foreground to the background. The compression increases with the scale up to a point where an inversion occurs and the compression starts decreasing. This zone of compression inversion is highlighted by the red circle.



(a) Input (b) [LJWF12] (c) Our result

Figure 26: **Guide extraction comparison.** The light guide extracted by [LJWF12] (b) is blurry because of the window of analysis. Our luminance guide (c) preserves the sharp transitions of the input thanks to the edge-aware descriptor filtering step.

input has a larger range of texture variations than the reference, we use those guides to deform the reference patches to better match the input. We also proposed a simple method to adapt a coarse alpha mask to a complex silhouette during the texture synthesis process in order to better preserve micro-structures. Our approach suffers from the following drawbacks:

- Our texture variations estimation from descriptor variations is based on the hypothesis that the texture is isotropic and uniform. Textures that do not satisfy this hypothesis may not be accurately described by our texture description.
- Our texture variations estimation is limited to luminance, compression and direction. This is efficient to represent simple variations of the textured object shape and environment, but may not accurately describe complex shape or environment variations. An example of such a failure case is shown in Figure 27 where the complex shape of the sheet is not well captured by our descriptors.
- Our mask expansion approach is simply based on color differences. This can be problematic when the texture micro-structures are overlapping or when only some part of the micro-structures should be expanded (the leaves but not the ground in between them for example). Semantic information would be helpful in those cases.

Despite those limitations, our algorithm is able to efficiently transfer textures between natural images in various cases. This algorithm could be extended to apply style transfer between images, based on the hypothesis that the style can be transferred through texture changes. On the other hand, it could also be combined with the color transfer approach presented in [AVHT17] in order to apply appearance transfer, as in [SL16]. Intuitions for these approaches using the algorithm presented in the report are given below.

9.1 Appearance transfer

Appearance transfer, defined as a combination of texture and color transfer as in [OVB*15, SL16], could be done with a combination of our color and texture transfers. If we place ourselves in a similar context as [SL16], we could start from a pair of input and reference images. Then, from texture descriptors computed with our filtering on the input and reference, find which regions

Input Reference Result

Figure 27: **Failure case.** In this case, even with luminance, scale and rotation alignments, we fail to match the complex shape deformation of the input sheet in the result.

Input Reference Result

Figure 28: **Artificial appearance transfer test.** The texture of the dirt road in the reference is synthesized on the input road, while color transfer is applied on the rest of the input image.

of the input have no similar regions in the reference and mark those as regions to re-synthesize. Color could be transferred between similar regions of the input and reference, while unmatched regions of the reference are used as exemplar for the input regions to re-synthesized. An example of result created with user-provided masks, combining our color and texture transfer framework, is shown in Figure 28. In this example, we synthesized dirt on the input road, while applying color transfer on the rest of the image. Textural information could be used to automatically compute the masks.

9.2 Style transfer

Style transfer methods usually try to reproduce the fine-scale textures of the reference, introduced by the medium used (brush strokes for example), while preserving the input image structure [FSDH16]. As such, our texture algorithm could be used for style transfer, using guides to define the input structure to preserve, and letting the synthesis reproduce the fine details of the reference. We show a test of this approach in Figure 29. In this example, we used the luminance guide to try and preserve the input structure. While the structure is mostly preserved, details

Input Reference Result

Figure 29: **Style transfer test with our texture transfer algorithm.** The luminance guide is used to try to preserve the input structure, while the texture synthesis reproduces the reference textural details. In this example, $\lambda_{\text{col}} = 1$, $\lambda_{\text{lum}} = 100$ and $\lambda_{\text{occ}} = 1$.

like the fence are lost. Moreover, synthesized textures are fairly repetitive. A straightforward solution could be to increase the occurrence weight, λ_{occ} in Section 5, however this introduces unwanted parts of the reference image in the result, such as sky parts on the road. The difficulty in this problem is that not all regions of the reference should be equally used, but rather only those that correspond to an input region. However, inside corresponding regions, we would like a uniform sampling of the reference in order to preserve the texture's complexity. Basically, we would need a local occurrence tracking rather than a global one. Additionally, adaptive patch size could also be a useful addition, as in [FSDH16].

References

[AVHT16] Arbelot B., Vergne R., Hurtut T., Thollot J.: Automatic texture guided color transfer and colorization. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2016), Expresive '16, Eurographics Association, pp. 21–32.

[AVHT17] Arbelot B., Vergne R., Hurtut T., Thollot J.: Local texture-based color transfer and colorization. *Computers & Graphics 62* (2017), 15–27.

[BCK*13] Bénard P., Cole F., Kass M., Mordatch I., Hegarty J., Senn M. S., Fleischer K., Pesare D., Breeden K.: Stylizing animation by example. *ACM Trans. Graph. 32*, 4 (July 2013), 119:1–119:12.

[BG87] Bigun J., Granlund G. H.: Optimal orientation detection of linear symmetry. In *Proceedings of the IEEE First International Conference on Computer Vision, London, Great Britain* (1987), pp. 433–438.

[BKCO16] Bellini R., Kleiman Y., Cohen-Or D.: Time-varying weathering in texture space. *ACM Trans. Graph. 35*, 4 (July 2016), 141:1–141:11.

[BPLD10] Bonneel N., Panne M. v. d., Lefebvre S., Drettakis G.: Proxy-Guided Texture Synthesis for Rendering Natural Scenes. In *Vision, Modeling, and Visualization (2010)* (2010), Koch R., Kolb A., Rezk-Salama C., (Eds.), The Eurographics Association.

[BSFG09] Barnes C., Shechtman E., Finkelstein A., Goldman D. B.: PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH) 28*, 3 (Aug. 2009).

[BSGF10] Barnes C., Shechtman E., Goldman D. B., Finkelstein A.: The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision* (Sept. 2010).

[BvdBL*06] Brox T., van den Boomgaard R., Lauze F., van de Weijer J., Weickert J., Mrázek P., Kornprobst P.: *Adaptive Structure Tensors and their Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 17–47.

[CVZ08] Cheng L., Vishwanathan S. V. N., Zhang X.: Consistent image analogies using semi-supervised learning. In *2008 IEEE Conference on Computer Vision and Pattern Recognition* (June 2008), pp. 1–8.

[DBP*15] Diamanti O., Barnes C., Paris S., Shechtman E., Sorkine-Hornung O.: Synthesis of complex image appearance from limited exemplars. *ACM Trans. Graph. 34*, 2 (Mar. 2015), 22:1–22:14.

[DSB*12] Darabi S., Shechtman E., Barnes C., Goldman D. B., Sen P.: Image melding: Combining inconsistent images using patch-based synthesis. *ACM Trans. Graph. 31*, 4 (July 2012), 82:1–82:10.

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer, 2001.

[EL99] Efros A., Leung T.: Texture synthesis by non-parametric sampling. In *In International Conference on Computer Vision* (1999), pp. 1033–1038.

[ELS08] Eisenacher C., Lefebvre S., Stamminger M.: Texture synthesis from photographs. In *Proceedings of the Eurographics conference* (2008).

[FH04] Fang H., Hart J. C.: Textureshop: Texture synthesis as a photograph editing tool. In *In Proc. SIGGRAPH 2004* (2004).

[FSDH16] Frigo O., Sabater N., Delon J., Hellier P.: Split and Match: Example-based Adaptive Patch Sampling for Unsupervised Style Transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Las Vegas, United States, June 2016).

[GEB15] Gatys L. A., Ecker A. S., Bethge M.: A neural algorithm of artistic style. *CoRR abs/1508.06576* (2015).

[HJO*01] Hertzmann A., Jacobs C. E., Oliver N., Curless B., Salesin D. H.: Image analogies. In *Proc. of the 28th annual conference on Computer graphics and interactive techniques* (2001), SIGGRAPH '01, pp. 327–340.

[HZW*06] Han J., Zhou K., Wei L.-Y., Gong M., Bao H., Zhang X., Guo B.: Fast example-based surface texture synthesis via discrete optimization. *Vis. Comput. 22*, 9 (Sept. 2006), 918–925.

[IEKM16] Iizuka S., Endo Y., Kanamori Y., Mitani J.: Single image weathering via exemplar propagation. *Computer Graphics Forum (Proc. of Eurographics 2016)* (2016).

[JDA*11] Johnson M. K., Dale K., Avidan S., Pfister H., Freeman W. T., Matusik W.: Cg2real: Improving the realism of computer generated images using a large collection of photographs. *IEEE Transactions on Visualization and Computer Graphics 17*, 9 (Sept 2011), 1273–1285.

[JFA*15] Jamriška O., Fišer J., Asente P., Lu J., Shechtman E., Sýkora D.: Lazyfluids: Appearance transfer for fluid animations. *ACM Trans. Graph. 34*, 4 (July 2015), 92:1–92:10.

[KEBK05] Kwatra V., Essa I., Bobick A., Kwatra N.: Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3 (July 2005), 795–802.

[KFCO*07] Kopf J., Fu C.-W., Cohen-Or D., Deussen O., Lischinski D., Wong T.-T.: Solid texture synthesis from 2d exemplars. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM.

[KNL*15] Kaspar A., Neubert B., Lischinski D., Pauly M., Kopf J.: Self tuning texture optimization. *Comput. Graph. Forum 34*, 2 (May 2015), 349–359.

[KSE*03] Kwatra V., Schödl A., Essa I., Turk G., Bobick A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003* (2003), 277–286.

[LFA*15] Lukáč M., Fišer J., Asente P., Lu J., Shechtman E., Sýkora D.: Brushables: Example-based Edge-aware Directional Texture Painting. *Computer Graphics Forum 34*, 7 (2015), 257–268.

[LH06] Lefebvre S., Hoppe H.: Appearance-space texture synthesis. *ACM Trans. Graph. 25*, 3 (July 2006), 541–548.

[LJWF12] Liu X., Jiang L., Wong T. T., Fu C. W.: Statistical invariance for texture synthesis. *IEEE Transactions on Visualization and Computer Graphics 18*, 11 (Nov 2012), 1836–1848.

[LL12] Lasram A., Lefebvre S.: Parallel patch-based texture synthesis. In *High Performance Graphics conference proceedings* (2012).

[LLH04] Liu Y., Lin W.-C., Hays J.: Near-regular texture analysis and manipulation. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 368–376.

[LSA*16] Lockerman Y. D., Sauvage B., Allègre R., Dischler J.-M., Dorsey J., Rushmeier H.: Multi-scale label-map extraction for texture synthesis. *ACM Trans. Graph. 35*, 4 (July 2016), 140:1–140:12.

[LXDR13] Lockerman Y. D., Xue S., Dorsey J., Rushmeier H.: Creating Texture Exemplars from Unconstrained Images.

[Mai06] Maitre H.: *Cours Traitement des Images.* 2006, ch. Les textures.

[OVB*15] Okura F., Vanhoey K., Bousseau A., Efros A. A., Drettakis G.: Unifying color and texture transfer for predictive appearance manipulation. In *Proceedings of the 26th Eurographics Symposium on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2015), EGSR '15, Eurographics Association, pp. 53–63.

[PBK13] Park H., Byun H., Kim C.: Multi-exemplar inhomogeneous texture synthesis. *Computers & Graphics 37*, 1–2 (2013), 54–64.

[Pio13] Piotr Dollar L. Z.: Structured Forests for Fast Edge Detection. In *ICCV* (December 2013), International Conference on Computer Vision.

[RAF03] Rosales R., Achan K., Frey B.: Unsupervised image translation. In *Proceedings Ninth IEEE International Conference on Computer Vision* (Oct 2003), pp. 472–478 vol.1.

[RCOL09] Rosenberger A., Cohen-Or D., Lischinski D.: Layered shape synthesis: Automatic generation of control maps for non-stationary textures. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 107:1–107:9.

[RSK13] Ruiters R., Schwartz C., Klein R.: Example-based interpolation and synthesis of bidirectional texture functions. *Computer Graphics Forum (Proceedings of the Eurographics 2013) 32*, 2 (May 2013), 361–370.

[RT06] Rong G., Tan T.-S.: Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2006), I3D '06, ACM, pp. 109–116.

[SH95] Stalling D., Hege H.-C.: Fast and resolution independent line integral convolution. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 249–256.

[SL16] Song Z. C., Liu S. G.: Sufficient image appearance transfer combining color and texture. *IEEE Transactions on Multimedia PP*, 99 (2016), 1–1.

[SPB*14] Shih Y., Paris S., Barnes C., Freeman W. T., Durand F.: Style transfer for headshot portraits. *ACM Trans. Graph. 33*, 4 (July 2014), 148:1–148:14.

[SPDF13] Shih Y., Paris S., Durand F., Freeman W. T.: Data-driven hallucination of different times of day from a single outdoor photo. *ACM Trans. Graph. 32*, 6 (2013), 200:1–200:11.

[WLKT09] Wei L.-Y., Lefebvre S., Kwatra V., Turk G.: State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR* (2009), Eurographics Association.

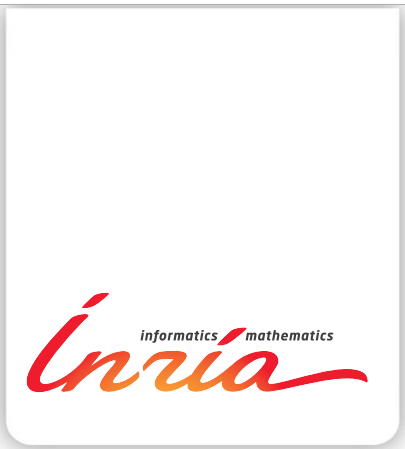
[WOBT09] Winnemöller H., Orzan A., Boissieux L., Thollot J.: Texture design and draping in 2d images. In *Proceedings of the Twentieth Eurographics Conference on Rendering* (Aire-la-Ville, Switzerland, Switzerland, 2009), EGSR'09, Eurographics Association, pp. 1091–1099.

[WSI07] Wexler Y., Shechtman E., Irani M.: Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell. 29*, 3 (Mar. 2007), 463–476.

[yWL00] yi Wei L., Levoy M.: Fast texture synthesis using tree-structured vector quantization. pp. 479–488.

[ZCC*13] Zhang W., Cao C., Chen S., Liu J., Tang X.: Style transfer via image component analysis. *Trans. Multi. 15*, 7 (Nov. 2013), 1594–1601.

[ZZV*03] Zhang J., Zhou K., Velho L., Guo B., Shum H.-Y.: Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22*, 3 (July 2003), 295–302.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399